Clipping in Neurocontrol by Adaptive Dynamic Programming

Michael Fairbank, *Student Member, IEEE*, Danil Prokhorov, *Senior Member, IEEE*, and Eduardo Alonso

***Abstract*—In adaptive dynamic programming, neurocontrol and reinforcement learning, the objective is for an agent to learn to choose actions so as to minimise a total cost function. In this paper we show that when discretized time is used to model the motion of the agent, it can be very important to do "clipping" on the motion of the agent in the final time step of the trajectory. By clipping we mean that the final time step of the trajectory is to be truncated such that the agent stops exactly at the first terminal state reached, and no distance further. We demonstrate that when clipping is omitted, learning performance can fail to reach the optimum; and when clipping is done properly, learning performance can improve significantly.**

The clipping problem we describe affects algorithms which use explicit derivatives of the model functions of the environment to calculate a learning gradient. These include Backpropagation Through Time for Control, and methods based on Dual Heuristic Programming. However the clipping problem does not significantly affect methods based on Heuristic Dynamic Programming, Temporal Difference or Policy-Gradient Learning algorithms.

I. Introduction

In Adaptive Dynamic Programming (ADP) [1], Neurocontrol [2], [3], and Reinforcement Learning (RL) [4], [5], an agent moves in a state space $\mathbb{S} \subset \mathbb{R}^n$, such that at integer time step t, it has state vector $\vec{x}_t \in \mathbb{S}$. $\mathbb{T}$ is a fixed set of *terminal states*, with $\mathbb{T} \subset \mathbb{S}$. At each time t, the agent chooses an action $\vec{u}_t$ which takes it to the next state according to the environment's model function

$$\vec{x}_{t+1} = f(\vec{x}_t, \vec{u}_t), \tag{1}$$

thus the agent passes through a trajectory of states $(\vec{x}_0, \vec{x}_1, \vec{x}_2, \ldots)$, terminating only when (and if) a terminal state is reached, as illustrated in Fig. 1. As shown in this figure, clipping is the concept of calculating the exact fraction in the final time step at which a boundary of terminal states is reached, and stopping the agent exactly at this boundary. The name clipping is taken by analogy to the concept in computer graphics. Without clipping, the discretization of time would cause the agent to penetrate slightly beyond the terminal boundary, as shown in the figure.

On transitioning from each state $\vec{x}_t$ to the next, the agent receives an immediate scalar cost U_t from the environment according to the function

$$U_t := U(\vec{x}_t, \vec{u}_t). \tag{2}$$

In addition, if the agent reaches a terminal state $\vec{x} \in \mathbb{T}$, then an additional terminal cost is given by the scalar function $\Phi(\vec{x})$.

M. Fairbank and E. Alonso are with the Department of Computer Science, School of Informatics, City University London, London, UK
D. Prokhorov is with Toyota Research Institute NA, Ann Arbor, Michigan
Manuscript received June 2013

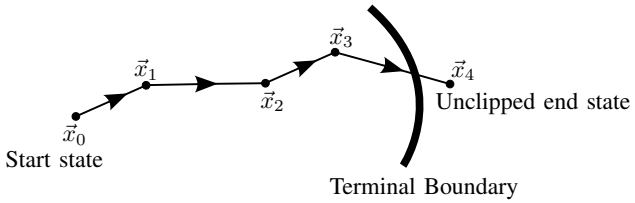


Fig. 1: A trajectory reaching a terminal state. The thick curved line indicates a boundary of terminal states. In this diagram, clipping does not take place, and the trajectory penetrates beyond the terminal boundary. When clipping is used correctly, we intend to stop the agent exactly at the point of intersection between the trajectory and the terminal boundary.

Throughout this paper, subscripts on variables will be used to indicate the time step of a trajectory. And from now on in the paper, we will only consider episodic, or finite horizon, environments; that is environments where all trajectories are guaranteed to meet a terminal state eventually.

The ADP problem is for the agent to learn to choose actions so as to minimise the expectation of the total long-term cost received from any given start state $\vec{x}_0$. Specifically, the problem is to find an *action network* $A(\vec{x}, \vec{z})$, where $\vec{z}$ is the parameter vector of a function approximator, which calculates an action

$$\vec{u}_t = A(\vec{x}_t, \vec{z}) \tag{3}$$

to take for any given state $\vec{x}_t$, such that the following long-term cost is minimised:

$$J(\vec{x}_0, \vec{z}) := \left\langle \sum_{t=0}^{T-1} \gamma^t U_t + \gamma^T \Phi(\vec{x}_T) \right\rangle \tag{4}$$

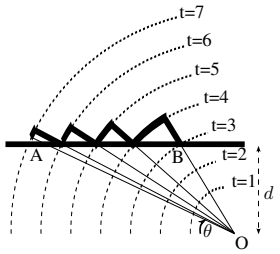
subject to (1), (2) and (3); where T is the time step at which the first terminal state is reached (which in general will be dependent on $\vec{x}_0$ and $\vec{z}$), where $\gamma \in [0, 1]$ is a constant *discount factor* that specifies the relative importance of long-term costs over short term ones, and where $\langle \cdot \rangle$ denotes expectation.

The function $J(\vec{x}_0, \vec{z})$ is called the *cost-to-go* function from state $\vec{x}_0$, or the *value function*.

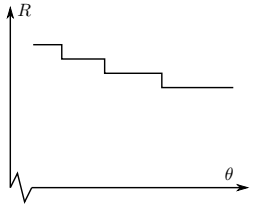
In this paper we show that when a large final impulse of cost $\Phi(\vec{x})$ is given at a terminal state $\vec{x} \in \mathbb{T}$, then failure to do clipping in the final time step of the trajectory can very significantly distort the direction of the learning gradient used by certain ADP algorithms, and thus prevent successful solution of the ADP problem. We also show that this problem is not lessened by sampling the time steps of the underlying continuous-time process at a higher rate. This

problem affects the commonly used ADP algorithms Dual Heuristic Programming (DHP) [6], [7], and Backpropagation Through Time (BPTT) [8], both of which are described in Section II, plus algorithms based on DHP such as Value-Gradient Learning [9], [10], [11]. These algorithms are all very closely related to each other [12], [13], and for purposes of explaining clipping as clearly as possible, we will use BPTT as the example.

BPTT works by calculating the quantity $\frac{\partial J}{\partial \vec{z}}$ directly and very efficiently for each trajectory sampled, enabling gradient descent to be performed on J with respect to $\vec{z}$. However if clipping is omitted then the gradient that BPTT calculates can be distorted enough to prevent learning. Fig. 2 illustrates the problems that arise without clipping.



(a) Spurious zigzag gradients can occur when clipping is not used.



(b) The graph of R versus θ yields no useful local gradient information. Hence minimising R with respect to θ using only $dR/d\theta$ would be impossible.

Fig. 2: An example of the problems that can occur when clipping is not used.

In Fig. 2a the agent starts at O and travels in a straight line at a constant speed, along a fixed chosen initial angle, θ. The straight line AB is a terminal boundary (i.e. a continuous line of states in $\mathbb{T}$). The dotted arcs represent the integer time steps that the agent passes through. If clipping is not used then the agent will stop on the first integer time step (i.e. on the first dotted arc) after passing the terminal boundary. This means the agent will finally stop at a point somewhere on the bold zigzag path from A to B. In Fig. 2b we see how the distance the agent travelled before stopping (R) varies with θ. If the cost-to-go function J was defined to be the total distance travelled before termination (i.e. if $J := R$), and the parameter vector of J was defined to be θ, then the ADP objective would be to minimise R with respect to θ. But Fig. 2b shows that there is no useful gradient information for learning, since $\frac{\partial J}{\partial \theta} = \frac{\partial R}{\partial \theta} = 0$, whenever it exists, and hence gradient descent on J with respect to θ would fail without clipping.

Situations can get even worse than this: In Fig. 3 we show a pathological example where the gradient of the graph is always in the opposite direction of the global minimum of R. This could occur for example if we were trying to minimise the function $J := R + y$ with respect to θ, for the situation in Fig. 2a, where y is the final y-coordinate of the agent, and R is the distance travelled before stopping.

In general, increasing the sampling rate of the discretization of time will not solve the problem, since that would simply make the dotted arcs in Fig. 2a squeeze closer together, and will make the teeth of the saw-tooth blade shape in Fig. 3 finer.

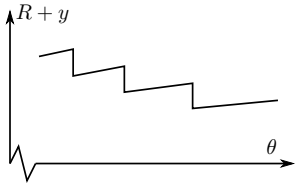


Fig. 3: A pathological example: Local gradient is opposite to global gradient.

The gradients in Figs. 2b and 3 would still not be helpful for learning.

We show how to solve the problem by incorporating clipping into the model and cost functions, $f(\vec{x}, \vec{u})$ and $U(\vec{x}, \vec{u})$, when terminal states are reached. BPTT and DHP make intensive use of the derivatives of these two functions, and hence we must carefully differentiate through the clipped versions of these functions. This is the important step that we derive in this paper, and this step corrects the gradient $\frac{\partial J}{\partial \vec{z}}$ to make it suitable for learning, and solves the problems explained by Fig. 2 and Fig. 3.

As well as terminal boundaries in state space that deliver impulses of cost, similar corrections would need making in environments where the model and cost functions change their behaviour discontinuously as the agent traverses a given continuous boundary in state space. These boundaries would act as refraction layers do to photons. As the agent crosses them, the learning gradient $\frac{\partial J}{\partial \vec{z}}$ would get twisted. The solution to this problem is similar to the one we propose for terminal boundaries, but we do not consider these non-terminal refraction layers any more in this paper.

The necessity for clipping affects any algorithm which calculates the derivatives of the model function, i.e. $\frac{\partial f}{\partial \vec{x}}$ directly, and when terminal states that deliver impulses of cost are present. For example the RL method of [14], which implements a continuous-time numerical differentiation to evaluate $\frac{\partial J}{\partial \vec{z}}$, will also be affected by this clipping problem. Likewise, the ADP methods of BPTT, DHP, GDHP [15] and Value-Gradient Learning are also affected by the requirement for clipping.

Clipping is not necessary for any problem where the termination condition is simply when a fixed integer number of time steps is reached, as we discuss further in Section III-D. Also our experiments in this paper show that the ADP algorithm called Heuristic Dynamic Programming (HDP, [6], [1], [7]) does not need clipping, since this algorithm does not make significant use of the derivatives of the model function. For the same reasons, the Policy-Gradient Learning methods of [16], [17] do not require clipping either. We discuss Policy-Gradient methods in Section V.

In the rest of this paper, in Section II we describe the affected ADP algorithms for control problems. In Section III we describe how to do the clipping and differentiate through the modified model functions, as is required for effective gradient descent. In Section IV we give experimental details of neural-network control problems, both with and without clipping. One of these problems is the classic cart-pole benchmark problem which we formulate in a way that

would be impossible for DHP to solve without clipping, and we show that the clipping methods enable us to solve this problem efficiently. In Section V we describe Policy-Gradient Learning methods and discuss why they don't require clipping, despite the methods' similarity to BPTT. That section also reviews the pros and cons between BPTT and Policy-Gradient methods. Finally, in Section VI, we give conclusions.

II. The ADP/RL Learning Algorithms

We describe three main ADP/RL algorithms first in their forms without clipping. The modifications necessary for clipping will be given in Section III.

A. Backpropagation Through Time for Control

Backpropagation Through Time (BPTT) can be applied to control problems, as described by [8]. In this section we derive and describe the algorithm. This is an algorithm that requires clipping in the environments we consider in this paper.

BPTT is an efficient algorithm to calculate $\frac{\partial J}{\partial \vec{z}}$ for a given trajectory. The combination of the BPTT gradient calculation with a gradient descent weight update can be used to solve control problems, i.e. by the weight update $\Delta \vec{z} = -\alpha \frac{\partial J}{\partial \vec{z}}$ for some small positive learning rate α.

Throughout this paper we make a notational convention that all vectors are columns, and differentiation of a scalar by a vector gives a column vector (e.g. $\frac{\partial J}{\partial \vec{x}}$ is a column). We define differentiation of a vector function by a vector argument as the transpose of the usual Jacobian notation. For example, $\frac{\partial A(\vec{x},\vec{z})}{\partial \vec{x}}$ $\frac{\partial f}{\partial \vec{x}}$ is a matrix with element (i,j) equal to $\frac{\partial A^j}{\partial \vec{x}^i}$. Similarly, $\frac{\partial f}{\partial \vec{x}}$ is the matrix with element $\left(\frac{\partial f}{\partial \vec{x}}\right)^{ij} = \frac{\partial f^j}{\partial \vec{x}^i}$.

Parentheses subscripted with a "t" are what we call *trajectory-shorthand notation*, which we define to indicate that a quantity is evaluated at time step t of a trajectory. For example $\left(\frac{\partial U}{\partial \vec{u}}\right)_t$ is shorthand for the function $\frac{\partial U(\vec{x},\vec{u})}{\partial \vec{u}}$ evaluated at $(\vec{x}_t, \vec{u}_t)$. Similarly, $\left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1} := \left.\frac{\partial J(\vec{x},\vec{z})}{\partial \vec{x}}\right|_{(\vec{x}_{t+1},\vec{z})}$, and $\left(\frac{\partial A}{\partial \vec{z}}\right)_t := \left.\frac{\partial A(\vec{x},\vec{z})}{\partial \vec{z}}\right|_{(\vec{x}_t,\vec{z})}$

For any given trajectory starting at state $\vec{x}_0$, the function $J(\vec{x}_0, \vec{z})$ given by (4) can be written recursively using equations (1)-(3), as:

$$J(\vec{x},\vec{z}) := U(\vec{x}, A(\vec{x},\vec{z})) + \gamma J(f(\vec{x}, A(\vec{x},\vec{z})),\vec{z}) \qquad (5)$$

with $J(\vec{x}_T,\vec{z}) := \Phi(\vec{x}_T)$ at the trajectory's terminal state, $\vec{x}_T \in \mathbb{T}$.

Differentiating (5) with respect to $\vec{z}$, and applying the chain rule, gives:

$$\begin{aligned}
&\left(\frac{\partial J}{\partial \vec{z}}\right)_t \\
&= \left(\frac{\partial}{\partial \vec{z}}(U(\vec{x}, A(\vec{x},\vec{z})) + \gamma J(f(\vec{x}, A(\vec{x},\vec{z})),\vec{z}))\right)_t \qquad \text{by (5)} \\
&= \left(\frac{\partial A}{\partial \vec{z}}\right)_t \left(\left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1}\right) + \gamma \left(\frac{\partial J}{\partial \vec{z}}\right)_{t+1}
\end{aligned}$$

where we used the chain rule, equations (1)-(3) and trajectory-shorthand notation. In this equation there are implied matrix-vector products that make use of the matrix notation defined above.

Expanding this recursion gives:

$$\left(\frac{\partial J}{\partial \vec{z}}\right)_0 = \sum_{t\geq 0} \gamma^t \left(\frac{\partial A}{\partial \vec{z}}\right)_t \left(\left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1}\right) \qquad (6)$$

This equation refers to the quantity $\frac{\partial J}{\partial \vec{x}}$ which can be found recursively by differentiating (5) with respect to $\vec{x}$, and using the chain rule, giving:

$$\begin{aligned}
\left(\frac{\partial J}{\partial \vec{x}}\right)_t &= \left(\frac{\partial U}{\partial \vec{x}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{x}}\right)_t \left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1} \\
&\quad + \left(\frac{\partial A}{\partial \vec{x}}\right)_t \left(\left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1}\right)
\end{aligned} \qquad (7)$$

with

$$\left(\frac{\partial J}{\partial \vec{x}}\right)_T = \left(\frac{\partial \Phi}{\partial \vec{x}}\right)_T \qquad (8)$$

at the terminal state, $\vec{x}_T \in \mathbb{T}$.

Equation (7) can be understood to be back-propagating the quantity $\left(\frac{\partial J}{\partial \vec{x}}\right)_{t+1}$ through the action network, model and cost functions to obtain $\left(\frac{\partial J}{\partial \vec{x}}\right)_t$, and giving the algorithm its name. Pseudocode for the whole BPTT algorithm is given in Alg. 1, where lines 2, 6 and 7 of the algorithm come from equations (8), (6) and (7) respectively. In the algorithm, the vector $\vec{p}$ holds the backpropagated value for $\frac{\partial J}{\partial \vec{x}}$. Q_x and Q_u are the derivatives of the Q-function with respect to $\vec{x}$ and $\vec{u}$ respectively, where the Q-function is defined by

$$Q(\vec{x},\vec{u},\vec{z}) := U(\vec{x},\vec{u}) + \gamma J(f(\vec{x},\vec{u}),\vec{z}).$$

The Q-function is a model based version of the Q-function defined in Q-learning [18]. It is similar to the cost-to-go function's recursive definition (5), but it differs in that it allows the first action chosen to be independent of the action network. This will be useful in deriving the clipping equations in Section III, but for now Q_x and Q_u can just be treated as internal variables in Alg. 1. The BPTT algorithm runs in time $O(\dim(\vec{z}))$ per trajectory step.

Algorithm 1 Backpropagation Through Time for Control.

Require: Trajectory calculated by (1) and (3).

1: $\frac{\partial J}{\partial \vec{z}} \leftarrow \vec{0}$
2: $\vec{p} \leftarrow \left(\frac{\partial \Phi}{\partial \vec{x}}\right)_T$
3: **for** $t = T-1$ to 0 step -1 **do**
4: $\quad Q_x \leftarrow \left(\frac{\partial U}{\partial \vec{x}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{x}}\right)_t \vec{p}$
5: $\quad Q_u \leftarrow \left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \vec{p}$
6: $\quad \frac{\partial J}{\partial \vec{z}} \leftarrow \frac{\partial J}{\partial \vec{z}} + \gamma^t \left(\frac{\partial A}{\partial \vec{z}}\right)_t Q_u$
7: $\quad \vec{p} \leftarrow Q_x + \left(\frac{\partial A}{\partial \vec{x}}\right)_t Q_u$
8: **end for**
9: $\vec{z} \leftarrow \vec{z} - \alpha \frac{\partial J}{\partial \vec{z}}$

B. Dual Heuristic Programming (DHP) and Heuristic Dynamic Programming (HDP)

Dual Heuristic Programming (DHP) and Heuristic Dynamic Programming (HDP) are ADP algorithms which use a critic function, and can require clipping in the environments we consider in this paper. Both of these algorithms were originally by Werbos [6] and are described more recently by [7], [19], [1], and we define them briefly here.

The use of critic functions allows these two algorithms to apply their learning rule on-line, unlike the previously described BPTT which needed to wait until a trajectory was completed before it could apply the learning weight update. DHP makes use of a *vector-critic* function $\widetilde{G}(\vec{x}, \vec{w})$ which produces a vector output of dimension $\mathbb{R}^{\dim(\vec{x})}$. This could be the output of a neural network with weight vector $\vec{w}$ and $\dim(\vec{x})$ inputs and outputs. The DHP weight update attempts to make the function $\widetilde{G}(\vec{x}, \vec{w})$ learn to output the gradient $\frac{\partial J}{\partial \vec{x}}$. HDP uses a *scalar-critic* function $\widetilde{V}(\vec{x}, \vec{w})$ which produces a scalar output. This could be the output of a neural network with weight vector $\vec{w}$ and $\dim(\vec{x})$ inputs, and just one output node. The HDP weight update attempts to make the function $\widetilde{V}(\vec{x}, \vec{w})$ learn to output the function $J(\vec{x}, \vec{z})$ for all $\vec{x} \in \mathbb{S}$. HDP is equivalent to the algorithm "TD(0)" from the RL literature [20].

Pseudocode for DHP is given in Alg. 2. Line 9 of the algorithm trains the critic with a learning rate $\beta > 0$, and line 10 implements a commonly used actor weight update described by [7] (using a learning rate $\alpha > 0$). The algorithm uses the same matrix notation for Jacobians and trajectory-shorthand notation as described in Section II-A, so that for example $\left(\frac{\partial \widetilde{G}}{\partial \vec{w}}\right)_t$ is the function $\frac{\partial \widetilde{G}}{\partial \vec{w}}$ evaluated at $(\vec{x}_t, \vec{w})$.

Pseudocode for HDP is given in Alg. 3. Lines 8 and 9 give the critic and action-network weight updates, respectively. Again the action-network weight update is the one described by [7], but model-free alternatives which don't require knowledge of the derivatives of f are also possible (e.g. [4, ch.6.6], or [21, sec 4.2]).

Backpropagation ([22], [23]) can be used to efficiently calculate $\frac{\partial \widetilde{V}}{\partial \vec{w}}$, $\frac{\partial \widetilde{V}}{\partial \vec{x}}$ and the products involving $\frac{\partial A}{\partial \vec{z}}$ and $\frac{\partial \widetilde{G}}{\partial \vec{w}}$. Using this method, both DHP and HDP can be implemented in a running time of $O(n)$ operations per time step of the trajectory, where $n = \max(\dim(\vec{w}), \dim(\vec{z}))$.

Algorithm 2 DHP with a Critic Network $\widetilde{G}(\vec{x}, \vec{w})$ and Action Network $A(\vec{x}, \vec{z})$.

1: $t \leftarrow 0$
2: **while** $\vec{x}_t \notin \mathbb{T}$ **do**
3: $\quad \vec{u}_t \leftarrow A(\vec{x}_t, \vec{z})$
4: $\quad \vec{x}_{t+1} \leftarrow f(\vec{x}_t, \vec{u}_t)$
5: $\quad \vec{p} \leftarrow \widetilde{G}(\vec{x}_{t+1}, \vec{w})$
6: $\quad Q_x \leftarrow \left(\frac{\partial U}{\partial \vec{x}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{x}}\right)_t \vec{p}$
7: $\quad Q_u \leftarrow \left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \vec{p}$
8: $\quad \vec{e} \leftarrow Q_x + \left(\frac{\partial A}{\partial \vec{x}}\right)_t Q_u - \widetilde{G}(\vec{x}_t, \vec{w})$
9: $\quad \vec{w} \leftarrow \vec{w} + \beta \left(\frac{\partial \widetilde{G}}{\partial \vec{w}}\right)_t \vec{e}$ {Critic network update}
10: $\quad \vec{z} \leftarrow \vec{z} - \alpha \left(\frac{\partial A}{\partial \vec{z}}\right)_t Q_u$ {Action network update}
11: $\quad t \leftarrow t + 1$
12: **end while**
13: $\vec{e} \leftarrow \left(\frac{\partial \Phi}{\partial \vec{x}}\right)_t - \widetilde{G}(\vec{x}_t, \vec{w})$
14: $\vec{w} \leftarrow \vec{w} + \beta \left(\frac{\partial \widetilde{G}}{\partial \vec{w}}\right)_t \vec{e}$ {Final critic update}

Algorithm 3 HDP with a Critic Network $\widetilde{V}(\vec{x}, \vec{w})$ and Action Network $A(\vec{x}, \vec{z})$.

1: $t \leftarrow 0$
2: **while** $\vec{x}_t \notin \mathbb{T}$ **do**
3: $\quad s \leftarrow 1$
4: $\quad \vec{u}_t \leftarrow A(\vec{x}_t, \vec{z})$
5: $\quad \vec{x}_{t+1} \leftarrow f(\vec{x}_t, \vec{u}_t)$
6: $\quad \vec{p} \leftarrow \left(\frac{\partial \widetilde{V}}{\partial \vec{x}}\right)_{t+1}$
7: $\quad Q_u \leftarrow \left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma \left(\frac{\partial f}{\partial \vec{u}}\right)_t \vec{p}$
8: $\quad \vec{w} \leftarrow \vec{w} + \beta \left(\frac{\partial \widetilde{V}}{\partial \vec{w}}\right)_t \left(sU(\vec{x}_t, \vec{u}_t) + \gamma \widetilde{V}(\vec{x}_{t+1}, \vec{w}) - \widetilde{V}(\vec{x}_t, \vec{w})\right)$ {Critic network update}
9: $\quad \vec{z} \leftarrow \vec{z} - \alpha \left(\frac{\partial A}{\partial \vec{z}}\right)_t Q_u$ {Action network update}
10: $\quad t \leftarrow t + 1$
11: **end while**
12: $\vec{w} \leftarrow \vec{w} + \beta \left(\frac{\partial \widetilde{V}}{\partial \vec{w}}\right)_t \left(\Phi(\vec{x}_t) - \widetilde{V}(\vec{x}_t, \vec{w})\right)$ {Final critic update}

III. Using and Differentiating Clipping in Learning

In this section we derive the formulae for the clipped model and cost functions, and their derivatives. We will denote the clipped versions of the original functions with a superscripted *C*, so that f^C, U^C and J^C will be the function names we use for the clipped versions of the model, cost and cost-to-go functions, respectively. The functions f^C and U^C are only defined for any state $\vec{x}_t$ that occurs immediately before a terminal state is reached, i.e. for which $\vec{x}_t \notin \mathbb{T}$ and for which $f(\vec{x}_t, \vec{u}_t) \in \mathbb{T}$.

These three clipped functions, f^C, U^C and J^C, are key concepts in this paper, because defining them clearly allows us to differentiate them carefully, and hence calculate the learning gradients correctly. This is what allows us to solve the clipping problem. Hence this section is the main contribution of this paper, in terms of implementation details for solving the clipping problem.

A. Calculation of the Clipped Model and Cost Functions

Suppose the agent is transitioning between states $\vec{x}_t$ and $f(\vec{x}_t, \vec{u}_t)$, and the state $f(\vec{x}_t, \vec{u}_t)$ would be beyond the terminal boundary unless clipping was applied. To calculate the clipping correctly, we imagine this state transition as occurring along the straight line segment from $\vec{x}_t$ to $f(\vec{x}_t, \vec{u}_t)$, i.e. the straight line given parametrically by position vector

$$\vec{r} = \vec{x}_t + s\vec{v}, \tag{9}$$

where

$$\vec{v} = f(\vec{x}_t, \vec{u}_t) - \vec{x}_t, \tag{10}$$

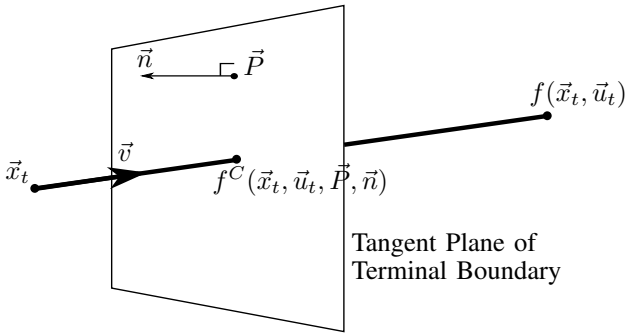


Fig. 4: The final state transition of a trajectory crossing the tangent plane of a terminal boundary. The unclipped line goes from $\vec{x}_t$ to $f(\vec{x}_t, \vec{u}_t)$. The line intersects the plane at a point given by the new *clipped* model function $f^C(\vec{x}_t, \vec{u}_t, \vec{P}, \vec{n})$.

and $s \in [0, 1]$ is a real parameter. This is illustrated in Fig. 4.

This straight line must intersect a boundary of terminal states. At the point of intersection, the tangent plane of the terminal boundary is given by $(\vec{r} - \vec{P}) \cdot \vec{n} = 0$ (i.e. where $\vec{r}$ is an arbitrary position vector that lies on a plane which has normal $\vec{n}$ and passes through a point with position vector $\vec{P}$, and where "$\cdot$" denotes the inner product between two vectors), as illustrated in Fig. 4. The constants $\vec{P}$ and $\vec{n}$ should be available from either the physical environment or from the collision-detection routine of the simulated environment.

At the intersection of the line and the plane, we have

$$\begin{aligned}&(\vec{x}_t + s\vec{v} - \vec{P}) \cdot \vec{n} = 0\\ \Rightarrow &s = \frac{(\vec{P} - \vec{x}_t) \cdot \vec{n}}{\vec{v} \cdot \vec{n}}.\end{aligned}$$

This value of s is a real number between 0 and 1 which indicates the fraction along the transition line from $\vec{x}_t$ to $f(\vec{x}_t, \vec{u}_t)$ at which the terminal boundary was encountered. We will refer to the value s as the "clipping fraction", and since it depends on $\vec{x}_t$, $\vec{u}_t$, $\vec{P}$ and $\vec{n}$, it is defined by the function:

$$s := S(\vec{x}_t, \vec{u}_t, \vec{P}, \vec{n}) := \frac{(\vec{P} - \vec{x}_t) \cdot \vec{n}}{(f(\vec{x}_t, \vec{u}_t) - \vec{x}_t) \cdot \vec{n}}. \quad (11)$$

Hence the clipped value of the final state is $\vec{x}_{t+1} = \vec{x}_t + S(\vec{x}_t, \vec{u}_t, \vec{P}, \vec{n})(f(\vec{x}_t, \vec{u}_t) - \vec{x}_t)$, which is found by combining equations (9), (10) and (11). This gives the function for the clipped model function as

$$f^C(\vec{x}, \vec{u}, \vec{P}, \vec{n}) := \vec{x} + S(\vec{x}, \vec{u}, \vec{P}, \vec{n})(f(\vec{x}, \vec{u}) - \vec{x}). \quad (12)$$

Assuming that "cost" is delivered at a uniform rate during the final state transition, the total clipped cost would be proportional to the clipping fraction, giving:

$$U^C(\vec{x}, \vec{u}, \vec{P}, \vec{n}) := S(\vec{x}, \vec{u}, \vec{P}, \vec{n})U(\vec{x}, \vec{u}). \quad (13)$$

Since the final clipped time step has duration $s \in [0, 1]$, the terminal cost $\Phi(\vec{x}_T)$ should only receive a discount of γ^s instead of the full discount γ. Hence, at the penultimate time step, $\vec{x}_{T-1}$, the total cost-to-go is

$$J^C(\vec{x}_{T-1}, \vec{z}) := U^C(\vec{x}_{T-1}, \vec{u}_{T-1}, \vec{P}, \vec{n}) + \gamma^s \Phi(\vec{x}_T). \quad (14)$$

Deciding to use γ^s in place of γ might seem like a trivial detail, but when differentiated, it provides useful information for the correct learning gradient, with clipping. This detail allows us to solve a version of the cart-pole benchmark problem, in Section IV-B, which would otherwise be impossible for DHP.

Alg. 4 illustrates how equations (1)-(3) and (11)-(14) would be used to evaluate a trajectory with clipping.

Algorithm 4 Unrolling a Trajectory with Clipping.

1: $t \leftarrow 0$, $J^C \leftarrow 0$
2: **while** $\vec{x}_t \notin \mathbb{T}$ **do**
3: $\quad \vec{u}_t \leftarrow A(\vec{x}_t, \vec{z})$
4: $\quad \vec{x}_{t+1} \leftarrow f(\vec{x}_t, \vec{u}_t)$
5: $\quad$ **if** $\vec{x}_{t+1} \in \mathbb{T}$ **then**
6: $\quad\quad$ Identify $\vec{P}$ and $\vec{n}$ by inspection of the intersection with the terminal boundary, $\mathbb{T}$.
7: $\quad\quad s \leftarrow S(\vec{x}_t, \vec{u}_t, \vec{P}, \vec{n})$ {using (11)}
8: $\quad\quad T \leftarrow t + 1$
9: $\quad\quad \vec{x}_T \leftarrow \vec{x}_t + s\,(\vec{x}_T - \vec{x}_t)$
10: $\quad\quad J^C \leftarrow J^C + (\gamma^t)\,(sU(\vec{x}_t, \vec{u}_t) + \gamma^s \Phi(\vec{x}_T))$
11: $\quad$ **else**
12: $\quad\quad J^C \leftarrow J^C + (\gamma^t)\,U(\vec{x}_t, \vec{u}_t)$
13: $\quad$ **end if**
14: $\quad t \leftarrow t + 1$
15: **end while**

Note that $\vec{P}$ and $\vec{n}$ are required by equations (11)-(13). These would be found during the collision-detection routine (i.e. line 6 of Alg. 4), from knowledge of the terminal-boundary orientation, together with knowledge of $\vec{x}_{T-1}$ and $f(\vec{x}_{T-1}, \vec{u}_{T-1})$. Knowledge of the orientation of the terminal boundary could come from a model of the physical environment's boundary; or if this model was not available, then a physical inspection of the actual boundary would need to take place. Examples of how these two vectors were found in our experiments are given in Sections IV-A and IV-B.

B. Calculation of the Derivatives of the Clipped Model and Cost Functions

The ADP algorithms described in Section II require the derivatives of the model function, and hence they will require the derivatives of the clipped model function $f^C(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ too. Fig. 5 shows how different the derivative of f^C can be from the derivative of f, and hence how important it is to get this correct in ADP/RL. This figure clarifies why algorithms that are dependent on $\frac{\partial f^C}{\partial \vec{x}}$ are critically affected by the need for clipping, and also that just reducing the duration of each time step tracking or simulating the motion will not solve the problem at all.

Differentiating the formula for $S(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ in (11) gives:

$$\begin{aligned}\frac{\partial S(\vec{x}, \vec{u}, \vec{P}, \vec{n})}{\partial \vec{x}} &= \frac{\partial}{\partial \vec{x}} \left(\frac{(\vec{P} - \vec{x}) \cdot \vec{n}}{(f(\vec{x}, \vec{u}) - \vec{x}) \cdot \vec{n}} \right) && \text{by (11)}\\ &= \frac{-\vec{n}}{\vec{v} \cdot \vec{n}} - \frac{(\vec{P} - \vec{x}) \cdot \vec{n}}{(\vec{v} \cdot \vec{n})^2} \frac{\partial (f(\vec{x}, \vec{u}) - \vec{x}) \cdot \vec{n}}{\partial \vec{x}} && \text{using (10)}\end{aligned}$$

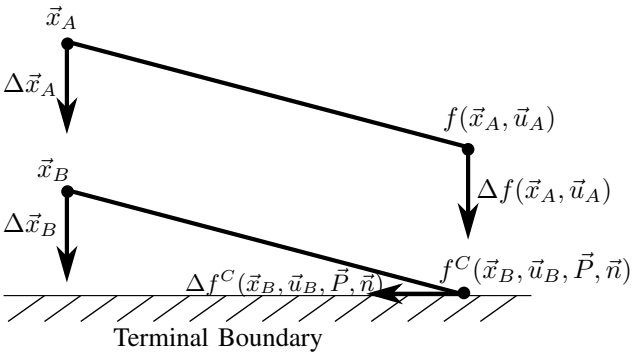


Fig. 5: This diagram shows how the derivatives of the model function $f(\vec{x}, \vec{u})$ radically change as the agent approaches a terminal boundary. The straight line segment from $\vec{x}_A$ to $f(\vec{x}_A, \vec{u}_A)$ represents a state transition that is not intersecting the terminal boundary. If the start of this line segment is perturbed in the direction of the arrow $\Delta\vec{x}_A$ then its other end will move in the direction indicated by the arrow $\Delta f(\vec{x}_A, \vec{u}_A)$. The line segment below, however, which starts at $\vec{x}_B$, does reach the terminal boundary. If the start of this line segment is moved in the direction of $\Delta\vec{x}_B$, then its end will move in a perpendicular direction, as indicated by the arrow $\Delta f^C(\vec{x}_B, \vec{u}_B, \vec{P}, \vec{n})$. This indicates that $\left(\frac{\partial f^C}{\partial \vec{x}}\right)_A$ is very different from $\left(\frac{\partial f}{\partial \vec{x}}\right)_B$, and hence this needs treating carefully in the ADP algorithms.

$$= \frac{-\vec{n}}{\vec{v}\cdot\vec{n}} - \frac{(\vec{P}-\vec{x})\cdot\vec{n}}{(\vec{v}\cdot\vec{n})^2}\left(\frac{\partial f}{\partial \vec{x}} - I\right)\vec{n} \qquad (15)$$

where I is the identity matrix, and the matrix notation is as defined in Section II-A. Similarly,

$$\frac{\partial S(\vec{x},\vec{u},\vec{P},\vec{n})}{\partial \vec{u}} = \frac{\partial}{\partial \vec{u}}\left(\frac{(\vec{P}-\vec{x})\cdot\vec{n}}{(f(\vec{x},\vec{u})-\vec{x})\cdot\vec{n}}\right) \quad \text{by (11)}$$

$$= -\frac{(\vec{P}-\vec{x})\cdot\vec{n}}{(\vec{v}\cdot\vec{n})^2}\frac{\partial(f(\vec{x},\vec{u})-\vec{x})\cdot\vec{n}}{\partial\vec{u}} \quad \text{using (10)}$$

$$= -\frac{(\vec{P}-\vec{x}_)\cdot\vec{n}}{(\vec{v}\cdot\vec{n})^2}\left(\frac{\partial f}{\partial \vec{u}}\right)\vec{n} \qquad (16)$$

Using these derivatives of $S(\vec{x},\vec{u},\vec{P},\vec{n})$, we can now differentiate the clipped model and cost functions, giving:

$$\frac{\partial f^C(\vec{x},\vec{u},\vec{P},\vec{n})}{\partial \vec{x}} = I + \frac{\partial S}{\partial \vec{x}}\vec{v}^T + s\left(\frac{\partial f}{\partial \vec{x}} - I\right) \quad \text{by (10)-(12)} \quad (17)$$

$$\frac{\partial f^C(\vec{x},\vec{u},\vec{P},\vec{n})}{\partial \vec{u}} = \frac{\partial S}{\partial \vec{u}}\vec{v}^T + s\frac{\partial f}{\partial \vec{u}} \quad \text{by (10)-(12)} \quad (18)$$

$$\frac{\partial U^C(\vec{x},\vec{u},\vec{P},\vec{n})}{\partial \vec{x}} = \frac{\partial S}{\partial \vec{x}}U(\vec{x},\vec{u}) + s\frac{\partial U}{\partial \vec{x}} \quad \text{by (13)} \quad (19)$$

$$\frac{\partial U^C(\vec{x},\vec{u},\vec{P},\vec{n})}{\partial \vec{u}} = \frac{\partial S}{\partial \vec{u}}U(\vec{x},\vec{u}) + s\frac{\partial U}{\partial \vec{u}} \quad \text{by (13)} \quad (20)$$

The cost-to-go function for the penultimate time step, equation (14), can be rewritten as a Q-function of both $\vec{x}$ and $\vec{u}$, to give

$$Q(\vec{x}_{T-1},\vec{u}_{T-1}) := U^C(\vec{x}_{T-1},\vec{u}_{T-1},\vec{P},\vec{n}) + \gamma^s\Phi(f^C(\vec{x}_{T-1},\vec{u}_{T-1},\vec{P},\vec{n})). \qquad (21)$$

Differentiating this with respect to $\vec{u}_{T-1}$ or $\vec{x}_{T-1}$ gives:

$$\left(\frac{\partial Q}{\partial \bullet}\right)_{T-1} = \left(\frac{\partial U^C}{\partial \bullet}\right)_{T-1} + \gamma^s\left(\left(\frac{\partial f^C}{\partial \bullet}\right)_{T-1}\left(\frac{\partial \Phi}{\partial \vec{x}}\right)_T + (\ln\gamma)\left(\frac{\partial S}{\partial \bullet}\right)_{T-1}\Phi(\vec{x}_T)\right) \qquad (22)$$

where $\bullet$ represents either $\vec{u}$ or $\vec{x}$.

This equation, which relies upon the derivatives of $f^C(\vec{x},\vec{u},\vec{P},\vec{n})$ and $U^C(\vec{x},\vec{u},\vec{P},\vec{n})$ (as defined in equations (15) to (20)), can be used to modify BPTT from Alg. 1 into its corresponding "with clipping" version given in Alg. 5. Equation (22) appears in the algorithm directly in lines 9-10.

The DHP and HDP algorithms need similar modifications to convert them to include clipping. Pseudocode for DHP with clipping is given in Alg. 6. In addition to those clipping modifications included in this algorithm, a further useful modification is line 4 which obviates the need for the "final critic update" line which appears in Alg. 2. This modification was included because $\widetilde{G}(\vec{x},\vec{w})$ can change discontinuously at the final time step of a trajectory, as indicated in Fig. 5, which would cause a small but unnecessary difficulty for learning by a smooth neural network.

Pseudocode for HDP with clipping can be generated by replacing the line "$\vec{x}_{t+1} \leftarrow f(\vec{x}_t, \vec{u}_t)$" of Alg. 3 by lines 4-13 of Alg. 4, and replacing the line that calculates Q_u by lines 5-14 of Alg. 5.

Algorithm 5 Backpropagation Through Time for Control, with Clipping.

1: Unroll full trajectory from start state $\vec{x}_0$ using Alg. 4, and retain the variables $\vec{x}_t$, $\vec{u}_t$, T, s, $\vec{P}$ and $\vec{n}$.
2: $\frac{\partial J}{\partial \vec{z}} \leftarrow \vec{0}$
3: $\vec{p} \leftarrow \left(\frac{\partial \Phi}{\partial \vec{x}}\right)_T$
4: **for** $t = T-1$ to 0 step -1 **do**
5: **if** $\vec{x}_{t+1} \in \mathbb{T}$ **then**
6: Calculate $\left(\frac{\partial S}{\partial \vec{x}}\right)_t$ and $\left(\frac{\partial S}{\partial \vec{u}}\right)_t$ by (15) and (16).
7: Calculate $\left(\frac{\partial f^C}{\partial \vec{x}}\right)_t$ and $\left(\frac{\partial f^C}{\partial \vec{u}}\right)_t$ by (17) and (18).
8: Calculate $\left(\frac{\partial U^C}{\partial \vec{x}}\right)_t$ and $\left(\frac{\partial U^C}{\partial \vec{u}}\right)_t$ by (19) and (20).
9: $Q_x \leftarrow \left(\frac{\partial U^C}{\partial \vec{x}}\right)_t + \gamma^s\left(\left(\frac{\partial f^C}{\partial \vec{x}}\right)_t\vec{p} + (\ln\gamma)\left(\frac{\partial S}{\partial \vec{x}}\right)_t\Phi(\vec{x}_T)\right)$
10: $Q_u \leftarrow \left(\frac{\partial U^C}{\partial \vec{u}}\right)_t + \gamma^s\left(\left(\frac{\partial f^C}{\partial \vec{u}}\right)_t\vec{p} + (\ln\gamma)\left(\frac{\partial S}{\partial \vec{u}}\right)_t\Phi(\vec{x}_T)\right)$
11: **else**
12: $Q_x \leftarrow \left(\frac{\partial U}{\partial \vec{x}}\right)_t + \gamma\left(\frac{\partial f}{\partial \vec{x}}\right)_t\vec{p}$
13: $Q_u \leftarrow \left(\frac{\partial U}{\partial \vec{u}}\right)_t + \gamma\left(\frac{\partial f}{\partial \vec{u}}\right)_t\vec{p}$
14: **end if**
15: $\frac{\partial J}{\partial \vec{z}} \leftarrow \frac{\partial J}{\partial \vec{z}} + \gamma^t\left(\frac{\partial A}{\partial \vec{z}}\right)_t Q_u$
16: $\vec{p} \leftarrow Q_x + \left(\frac{\partial A}{\partial \vec{x}}\right)_t Q_u$
17: **end for**
18: $\vec{z} \leftarrow \vec{z} - \alpha\frac{\partial J}{\partial \vec{z}}$

Algorithm 6 DHP with Clipping.

1: $t \leftarrow 0$
2: **while** $\vec{x}_t \notin \mathbb{T}$ **do**
3: Evaluate $\vec{x}_{t+1}$, with clipping, by lines 3-13 of Alg. 4.
4: $\vec{p} \leftarrow \begin{cases} \left(\frac{\partial \Phi}{\partial \vec{x}}\right)_{t+1} & \text{if } \vec{x}_{t+1} \in \mathbb{T} \\ \widetilde{G}(\vec{x}_{t+1}, \vec{w}) & \text{if } \vec{x}_{t+1} \notin \mathbb{T} \end{cases}$
5: Calculate Q_x and Q_u by lines 5-14 of Alg. 5.
6: $\vec{e} \leftarrow Q_x + \left(\frac{\partial A}{\partial \vec{x}}\right)_t Q_u - \widetilde{G}(\vec{x}_t, \vec{w})$
7: $\vec{w} \leftarrow \vec{w} + \beta \left(\frac{\partial \widetilde{G}}{\partial \vec{w}}\right)_t \vec{e}$ {Critic network update}
8: $\vec{z} \leftarrow \vec{z} - \alpha \left(\frac{\partial A}{\partial \vec{z}}\right)_t Q_u$ {Action network update}
9: $t \leftarrow t + 1$
10: **end while**

C. Implementing Clipping Efficiently and Correctly

To demonstrate how clipping would be correctly implemented with an ADP/RL algorithm, we use the BPTT algorithm for illustration. In an implementation of BPTT with clipping, we would first evaluate a trajectory by Alg. 4. During this stage, we would record the full trajectory $(\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_T)$ and actions $(\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_{T-1})$ and also, during the collision with the terminal boundary, we would record $\vec{P}$ and $\vec{n}$ and the clipping fraction, s. We then have enough information to be able to run the BPTT algorithm with clipping (Alg. 5).

To ensure the correctness of our implementations in each experiment and environment which we tackled, we first verified all of the derivatives of $S(\vec{x}, \vec{u}, \vec{P}, \vec{n})$, $f^C(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ and $U^C(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ numerically, with respect to both $\vec{x}$ and $\vec{u}$, at least a few times. When all of these derivatives were all satisfactorily programmed and checked, we then checked by numerical differentiation that the overall BPTT implementation was calculating the derivative $\frac{\partial J^C}{\partial \vec{z}}$ correctly.

For an example of the numerical differentiations used, the final check of BPTT was done by a central-differences numerical derivative for each component i of the weight vector $\vec{z}$, to verify that

$$\frac{\partial J^C}{\partial \vec{z}^i} = \frac{J^C(\vec{x}_0, \vec{z} + \epsilon \vec{e}_i) - J^C(\vec{x}_0, \vec{z} - \epsilon \vec{e}_i)}{2\epsilon} + O(\epsilon^2)$$

where ϵ is a small positive constant, and $\vec{e}_i$ is the ith Euclidean standard basis vector. In this verification equation, each $J^C(\cdot)$ term appearing in the right-hand side would be computed by executing Alg. 4 from the trajectory start point $\vec{x}_0$; and the theoretical value of $\frac{\partial J^C}{\partial \vec{z}}$ appearing in the left-hand side would be computed by Alg. 5.

In HDP and DHP, the derivatives of $S(\vec{x}, \vec{u}, \vec{P}, \vec{n})$, $f^C(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ and $U^C(\vec{x}, \vec{u}, \vec{P}, \vec{n})$ would be calculated and verified as above. However with HDP and DHP it is more difficult to check the overall critic weight updates numerically, since they are not true gradient descent on any analytic function [24]. For these algorithms, it is still possible to verify the key algorithmic modifications related to clipping, by just checking the derivatives of the Q-function given by (22). These derivatives can be compared to the numerical derivatives of (21) with respect to $\vec{x}$ and $\vec{u}$.

D. Clipping with Trajectories of Fixed or Variable Length

In situations where trajectories are of predetermined fixed length, clipping is not necessary. This is in contrast to the problems considered in the introduction, which were variable-length problems, since the trajectory lengths were determined by the environment (e.g. a trajectory terminates only when the agent crashes into a wall). In this section we will consider the difference between these two types of episodic problem, i.e. between fixed-length and variable-length problems. Only in variable-length problems is clipping necessary.

In the fixed-length problem, the clipping fraction defined by (11) is always $s \equiv 1$, and therefore $\frac{\partial S}{\partial \vec{x}} = \vec{0}$, $\frac{\partial S}{\partial \vec{u}} = \vec{0}$ and $\gamma^s = \gamma$. Hence the clipped model and cost functions are identical to their unclipped counterparts, and therefore it is not necessary to implement any program code specifically to handle clipping. This might be one reason why the need for clipping has not previously been noted in the research literature, since most episodic problems considered have been fixed-length.

However the fixed-length problem does have one minor different complication, in that it is often necessary to include the time step into the state vector. This is because the optimal actions and cost-to-go function will often be dependent upon the number of incomplete steps in a trajectory.

Of course for both fixed-length and variable-length problems, it is important to ensure the terminal cost function $\Phi(\vec{x})$ is learnt correctly by the learning algorithm. The pseudocode shows explicitly how to do this (e.g. for BPTT, see line 2 of Algs. 1 and 5. For DHP and HDP, see lines commented as "final critic update", and line 4 of Alg. 6.)

IV. Experimental Results

This section describes two neural-network based ADP/RL control problems which require clipping to be solved well.

In all experiments the action and critic networks used were multi-layer perceptrons (MLPs, see [25] for details). Each MLP had $\dim(\vec{x})$ input nodes, 2 hidden layers of 6 nodes each, and one output layer, with short-cut connections connecting all pairs of layers. The output layers were dimensioned as follows: Each action network had $\dim(\vec{u})$ output nodes; each HDP critic network had 1 output node; and each DHP critic had $\dim(\vec{x})$ output nodes. All network nodes had bias weights, as is usual in MLP architectures. The activation functions used were hyperbolic tangent functions, except for the critic network's output layer which was always a linear activation function (with linear slope as specified in the individual experiments, below). At the start of each experimental trial, neural weights were initialised randomly in the range $[-.1, .1]$, with uniform probability distribution.

A. Vertical-Lander problem

A spacecraft is dropped in a uniform gravitational field, and its objective is to make a fuel-efficient gentle landing. The spacecraft is constrained to move in a vertical line, and a

single thruster is available to make upward accelerations. The state vector $\vec{x} = (h, v, u)^T$ has three components: height (h), velocity (v), and fuel remaining (u). The action vector, a, is one-dimensional (so that $\vec{u} \equiv a \in \mathbb{R}$) producing accelerations $a \in [0, 1]$. The Euler method with time-step Δt is used to integrate the motion, giving model functions:

$$f((h, v, u)^T, a) := (h + v\Delta t, v + (a - k_g)\Delta t, (k_u)u - a\Delta t)^T$$
$$U((h, v, u)^T, a) := (k_f)a\Delta t \quad (23)$$

Here, $k_g = 0.2$ is a constant giving the acceleration due to gravity; the spacecraft can produce greater acceleration than that due to gravity. $k_f = 4$ is a constant giving fuel penalty. $k_u = 1$ is a unit conversion constant. We used $\Delta t = 1$ in our main experiments here.

Trajectories terminate as soon as the spacecraft hits the ground ($h = 0$) or runs out of fuel ($u = 0$). These two conditions define $\mathbb{T}$. This is a variable-length problem, and there is no need to use a discount factor, so we fixed $\gamma = 1$. On termination, the algorithms need to choose values for $\vec{P}$, and $\vec{n}$ which describe the orientation of the terminal-boundary tangent plane. These choices are given for this experiment in Table I. In the case that the final unclipped state transition crosses both terminal planes, then the one that is crossed first (i.e the one that produces a smaller clipping fraction by (11)) is to be used.

In addition to the cost function $U(\vec{x}, a)$ defined above, a final impulse of cost defined by,

$$\Phi(\vec{x}_T) := \frac{1}{2}mv^2 + m(k_g)h, \quad (24)$$

is given as soon as the lander reaches a terminal state, where $m = 2$ is the mass of the spacecraft. The two terms in the final impulse of cost are the kinetic and potential energy, respectively. The first cost term penalises landing too quickly. The second term is a cost term equivalent to the kinetic energy that the spacecraft would acquire by crashing to the ground under free fall (i.e. with $a = 0$), so to minimise this cost the spacecraft must learn to not run out of fuel.

The input vector to the action and critic networks was $\vec{x}' = (h/100, v/10, u/50)^T$, and the model and cost functions were redefined to act on this rescaled input vector directly. The action network's output y was rescaled to give the action by $A(\vec{x}, \vec{z}) := (y + 1)/2$ directly. We tested each algorithm in batch mode, operating on five trajectories simultaneously. Those five trajectories had fixed start points, which had been randomly chosen in the region $h \in (0, 100)$, $v \in (-10, 10)$ and $u = 30$.

Fig. 6 shows learning performance of the BPTT, DHP and HDP algorithms, both with and without clipping. Each graph shows five curves, and each curve shows the learning performance from a different random weight initialisation. The learning rates for the three algorithms were: BPTT ($\alpha = 0.01$); DHP ($\alpha = 0.001$, $\beta = 0.00001$); and HDP ($\alpha = 0.00001$, $\beta = 0.00001$). The critic-network's output layer's activation function had a linear slope of 20 in the DHP experiment and 10 in the HDP experiment. In BPTT and DHP, the true derivatives of equations (23)-(24) were used where needed.

TABLE I: Terminal Boundary Planes used in vertical-lander experiment. The state vector used here is $\vec{x} = (h, v, u)^T$.

Termination Condition Breached	Position Vector of Plane, $\vec{P}^T$	Normal Vector to Plane, $\vec{n}^T$
$h \leq 0$ (hits ground)	(0,0,0)	(1,0,0)
$u \leq 0$ (no fuel)	(0,0,0)	(0,0,1)

Because HDP is an algorithm which requires stochastic exploration to optimise the ADP/RL problem effectively [26], in the HDP experiment we had to modify (3) to choose exploratory actions. Hence for the HDP experiment we used

$$\vec{u}_t = A(\vec{x}_t, \vec{z}) + X_\sigma, \quad (25)$$

where X_σ is a normally distributed random variable with mean zero and standard deviation $\sigma = 0.1$.

These graphs show the clear stability and performance advantages of using clipping correctly for the BPTT and DHP algorithms. The graphs also confirm that the HDP algorithm is not significantly affected by the need for clipping. The reason that clipping is important for BPTT and DHP is illustrated in Fig. 8.

Fig. 7 shows that the need for clipping is not diminished just by using a smaller Δt value.

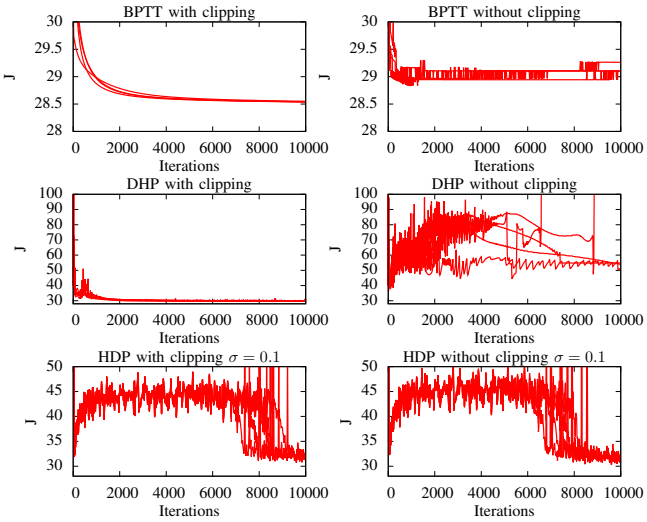


Fig. 6: Vertical-Lander solutions by BPTT, DHP and HDP using $\Delta t = 1$.

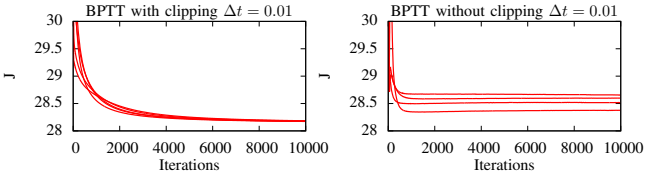


Fig. 7: Vertical Lander with $\Delta t = 0.01$.

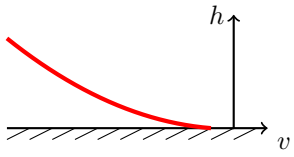


Fig. 8: The tail end of an optimal trajectory in the vertical-lander problem. As the spacecraft lands gently, it increases the velocity (v) as it approaches the ground ($h = 0$), hence making the curved trajectory shown. As the trajectory curve approaches the terminal boundary, clipping affects the gradient $\frac{\partial J^C}{\partial \vec{x}}$ as shown in Fig. 5, and hence this twists the gradient $\frac{\partial J^C}{\partial \vec{x}}$ discontinuously at the terminal state $\vec{x}_T$.

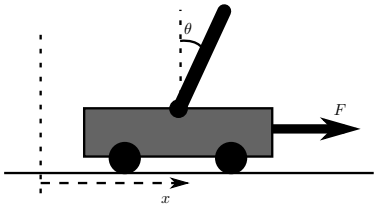


Fig. 9: Cart-pole benchmark problem. A pole with a pivot at its base is balancing on a cart. The objective is to apply a changing horizontal force F to the cart which will move the cart backwards and forwards so as to balance the pole vertically. State variables are pole angle, θ, and cart position, x, plus their derivatives with respect to time, $\dot{\theta}$ and $\dot{x}$.

B. Cart-Pole Experiment

We investigated the effects of clipping in the well known cart-pole benchmark problem described in Fig. 9. We considered the version of this problem used by [27], where the total trajectory cost is a function of the duration that the pole could be balanced for. Clearly, unless clipping is used properly, the duration will be an integer number of time steps, and since this is not smooth and differentiable, it will cause problems (become impossible) for DHP and BPTT. Hence traditionally when DHP or BPTT are used for the cart-pole problem, a different cost function would be used, one that is differentiable and proportional to the deviation from the balanced position (e.g. see [28]). However in this section we show that by using clipping, DHP and BPTT can be successful with the duration-based reward. Since it is not possible to do this without clipping, we assume this is the first published version of this solution by DHP/BPTT.

The equation of motion for the frictionless cart-pole system ([27], [29], [28]) is:

$$\ddot{\theta} = \frac{g \sin\theta - \cos\theta \left[\frac{F + ml\dot{\theta}^2 \sin\theta}{m_c + m}\right]}{l\left[\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}\right]} \tag{26}$$

$$\ddot{x} = \frac{F + ml\left[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right]}{m_c + m} \tag{27}$$

where gravitational acceleration, $g = 9.8ms^{-2}$; cart's mass, $m_c = 1kg$; pole's mass, $m = 0.1kg$; half pole length, $l = 0.5m$; $F \in [-10, 10]$ is the force applied to the cart, in Newtons; and the pole angle, θ, is measured in radians. The motion was integrated using the Euler method with a time constant $\Delta t = 0.02$, which, for a state vector $\vec{x} \equiv (x, \theta, \dot{x}, \dot{\theta})^T$, gives a model function $f(\vec{x}, \vec{u}) = \vec{x} + (\dot{x}, \dot{\theta}, \ddot{x}, \ddot{\theta})^T \Delta t$.

TABLE II: Terminal Boundary Planes used in cart-pole experiment. The state vector used here is $\vec{x} = (x, \dot{x}, \theta, \dot{\theta}, t)^T$.

Termination Condition Breached	Position Vector of Plane, $\vec{P}^T$	Normal Vector to Plane, $\vec{n}^T$
$\theta \geq \pi/15$	(0,0,π/15,0,0)	(0,0,−1,0,0)
$\theta \leq -\pi/15$	(0,0,−π/15,0,0)	(0,0,1,0,0)
$x \geq 2.4$	(2.4,0,0,0,0)	(−1,0,0,0,0)
$x \leq -2.4$	(−2.4,0,0,0,0)	(1,0,0,0,0)
$t \geq 300$	(0,0,0,0,300)	(0,0,0,0,1)

The pole motion continues until it reaches a terminal state or until the pole is successfully balanced for 300 time steps, i.e. 6 seconds of real time. Terminal states ($\mathbb{T}$) are defined to be any state with $|x| \geq 2.4$, or $|\theta| \geq \frac{\pi}{15}$ (i.e. 12 degrees), or $t \geq 300$. Termination plane constants are given in Table II.

The duration-based cost function of [27] is equivalent to

$$U(\vec{x}, u) := 0, \tag{28}$$

for non-terminal states, and,

$$\Phi(\vec{x}) := \begin{cases} 1 & \text{if } T < 300, \\ 0 & \text{otherwise,} \end{cases} \tag{29}$$

for terminal states $\vec{x} \in \mathbb{T}$. When the above two cost functions are used in conjunction with a discount factor $\gamma < 1$, and when the pole eventually falls over (i.e. when $T < 300$), the total trajectory cost is $J(\vec{x}_0, \vec{z}) \equiv \gamma^T$, where T is the time at which the trajectory terminated. Since this function decreases with T, minimising it will increase T, i.e. lead to successful pole balancing.[1]

We tested the three algorithms BPTT, DHP and HDP on this problem with a discount factor $\gamma = 0.97$. To ensure the state vector was suitably scaled for input to the MLPs, we used rescaled state vectors $\vec{x}'$ defined by $\vec{x}' = (0.16x, 15\theta/\pi, \dot{x}, 4\dot{\theta}, t/300)^T$, with θ in radians, throughout the implementation. As noted by [28], choosing an appropriate state-space scaling can be critical to successful convergence of actor-critic architectures in the cart-pole problem. Note that our implementation also uses the time step t as an input to the neural network, since the cost-to-go function that is being learned does depend upon t. The output of the action network, y, was multiplied by 10 to give the control force $F = A(\vec{x}, \vec{z}) := 10y$. The learning rates for the algorithms that we used were: BPTT ($\alpha = 0.1$); DHP ($\alpha = 0.001$, $\beta = 0.01$); HDP ($\alpha = 0.01$, $\beta = 0.1$). The DHP and HDP critics used a final-layer activation-function slope of 0.1. HDP used a policy exploration rate of $\sigma = 0.15$ (using (25)), and the other

[1]Previously, other researchers may have used $\Phi(\vec{x}) := 1$ instead of our equation (29), and may have stopped training the neural networks as soon as perfect balancing first occurs (e.g. [27]). We did not stop training like this, and therefore found that using (29) produced more stable results for HDP than the results when stopping training. It makes no difference to the performance of the DHP/BPTT algorithms.

algorithms used $\sigma = 0$. The exact derivatives of the model and cost functions were made available to the algorithms.

During learning, each trajectory was defined to start at the point $x = 0$, $\theta = 0$, $\dot{x} = 0.4$, $\dot{\theta} = 0$. This is a start state from which the pole will quickly topple over, unless corrective control actions are taken.

The performance of the three algorithms, both with and without clipping, are shown in Fig. 10. Each graph shows the balancing duration versus the training iteration, for an ensemble of five different curves, with each curve representing a training run from a different random weight initialisation.

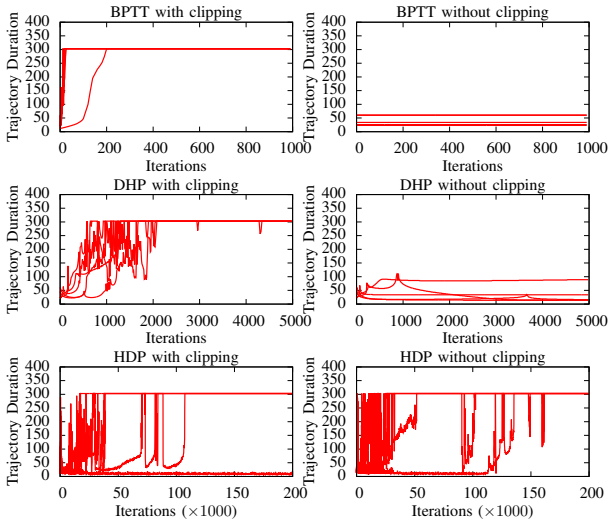


Fig. 10: Cart-pole solutions by BPTT, DHP and HDP.

The results show that using clipping correctly enables both the DHP and BPTT algorithms to solve this problem consistently, and without clipping it is impossible for both algorithms. The results show that HDP is largely unaffected by clipping.

This problem is interesting in that the cost functions defined by (28) and (29) would be completely inappropriate for learning with BPTT/DHP unless clipping is used correctly, since they have zero derivatives everywhere, i.e. $\left(\frac{\partial \Phi}{\partial \vec{x}}\right)_T \equiv \vec{0}$, $\frac{\partial U}{\partial \vec{u}} \equiv \vec{0}$ and $\frac{\partial U}{\partial \vec{x}} \equiv \vec{0}$. The clipping algorithm solves this problem through the expression in (22) given by $(\ln \gamma)\left(\frac{\partial S}{\partial \bullet}\right)_{T-1} \Phi(\vec{x}_T)$, which appears in lines 9-10 of Alg. 5. This expression allows for a useful learning gradient to be obtained. For it to work, we must have $\ln(\gamma) \neq 0$, which requires that $\gamma < 1$, and also we must have $\Phi(\vec{x}_T) \neq 0$. Furthermore, the derivatives of the clipping fraction, i.e. $\frac{\partial S}{\partial \vec{x}}$ and $\frac{\partial S}{\partial \vec{u}}$, must be calculated correctly by (15) and (16) for the pole-balancing problem to be solved.

V. A Note on Policy-Gradient Methods

As we have seen, BPTT is an algorithm which can be used for gradient descent on the total cost-to-go function $J(\vec{x}, \vec{z})$ with respect to $\vec{z}$. Another class of algorithms which do something similar are Policy-Gradient Learning (PGL) methods. These include the REINFORCE algorithm by [16], plus related methods (e.g. [17]). PGL methods are stochastic algorithms which do gradient descent of the form

$$\langle \Delta \vec{z} \rangle = -\alpha \frac{\partial \langle J \rangle}{\partial \vec{w}}. \tag{30}$$

Although these weight updates superficially look similar to the BPTT design, they do not use any explicit derivatives of the model or cost functions, and thus are not affected by the need for clipping. For example, the REINFORCE weight update is defined, for trajectories of length one, to be:

$$\begin{aligned} \Delta \vec{z} &= -\alpha \frac{\partial ln(g(\vec{u}_0|\vec{x}_0, \vec{z}))}{\partial \vec{z}} (U(\vec{x}_0, \vec{u}_0) - b) \\ &= -\alpha \frac{1}{g(\vec{u}_0|\vec{x}_0, \vec{z})} \frac{\partial g(\vec{u}_0|\vec{x}_0, \vec{z})}{\partial \vec{z}} (U(\vec{x}_0, \vec{u}_0) - b) \end{aligned} \tag{31}$$

where $\alpha > 0$ is small learning-rate constant, and b is a constant "baseline" scalar, and $g(\vec{u}_0|\vec{x}_0, \vec{z})$ is a probability distribution that forms the policy, such that action $\vec{u}_0$ is randomly sampled from this distribution, and the distribution $g(\vec{u}_0|\vec{x}_0, \vec{z})$ is modelled by a function approximator with weight vector $\vec{z}$ and input $\vec{x}_0$. Clearly this weight update (31) includes no derivatives of $f(\vec{x}, \vec{u})$, and hence has no need for clipping. However the expectation of this weight update is proven by [16] to be equivalent to (30), for any choice of the baseline constant.

The BPTT weight update is $\Delta \vec{z} = -\alpha \frac{\partial J}{\partial \vec{w}}$. In stochastic environments, the BPTT weight update would therefore average to

$$\langle \Delta \vec{z} \rangle = -\alpha \left\langle \frac{\partial J}{\partial \vec{w}} \right\rangle. \tag{32}$$

The derivation of the BPTT algorithm, in Section II-A, shows that this algorithm does require derivatives of $f(\vec{x}, \vec{u})$, and hence does require clipping.

So how do we reconcile that for two such similar algorithms, BPTT requires clipping, but PGL does not? The answer lies in the subtle difference between equations (30) and (32), i.e. the fact that in general, the derivative of a mean can be different from the mean of a derivative. In the PGL case, the $\langle J \rangle$ term has a blurring effect which first smooths out all of the jagged bumps in the J versus $\vec{z}$ graph (for example as shown in Fig. 2b), and then PGL performs gradient descent on this blurred-out graph. In contrast, BPTT first calculates the gradient of various randomly chosen points of this graph, and then averages out the answer, and clearly in the case of Fig. 2b, this approach will not work (unless clipping is done).

This shows that PGL methods have an advantage over BPTT methods in avoiding the need for clipping. However this complements the natural advantages that BPTT has over PGL, which are that BPTT can accumulate a learning gradient in just one trajectory, and therefore every single weight update provides useful learning. In contrast, PGL must form the mean from many weight updates before it can learn anything useful. In fact, a major area of research for PGL methods is to reduce the variance in these stochastic weight updates, so that the mean forms faster [17].

Other differences between the methods are that PGL is a model-free algorithm, whereas BPTT is model-based and requires knowledge of the function $f(\vec{x}, \vec{u})$ and its derivatives. A flip side of being model-based, is that when BPPT is used,

trajectories automatically bend themselves into locally optimal shapes, but the model-free PGL methods require explicit (and usually stochastic) exploration of the environment to achieve this.

To summarise, it is clear that the rival methods of BPTT and PGL have multiple relative pros and cons, and it is good to be aware of all of these issues.

VI. Conclusions

The problem of clipping for ADP/RL and neurocontrol algorithms has been demonstrated and motivated. Without clipping, algorithms which rely on the derivatives of the model and cost functions can fail to work. The solution is to apply clipping, and then to correctly differentiate the model and cost functions in the final time step. This solution has been given in the form of the equations, plus in the form of clear pseudocode for the two major affected ADP algorithms: DHP and BPTT.

Two neural-network experiments have confirmed the importance of applying clipping correctly. These included a cart-pole experiment, where clipping was found to be essential, and in a vertical-lander experiment, where clipping produced a significant improvement of performance.

The situations in which clipping is needed have been made clear, and those situations where it can be ignored have also been specified.

References

[1] F.-Y. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: An introduction," *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 39–47, 2009.

[2] P. J. Werbos, "Neurocontrol: Where it is going and why it is crucial," in *Artifical Neural Networks II (ICANN-2), North Holland, Amsterdam*, I. Aleksander and J. Taylor, Eds., 1992, pp. 61–68.

[3] M. Hagan and H. Demuth, "Neural networks for control," in *American Control Conference, 1999. Proceedings of the 1999*, vol. 3. IEEE, 1999, pp. 1642–1656.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachussetts, USA: The MIT Press, 1998.

[5] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *Control Systems, IEEE*, vol. 32, no. 6, pp. 76–105, 2012.

[6] P. J. Werbos, "Approximating dynamic programming for real-time control and neural modeling." in *Handbook of Intelligent Control*, White and Sofge, Eds. New York: Van Nostrand Reinhold, 1992, ch. 13, pp. 493–525.

[7] D. Prokhorov and D. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 997–1007, 1997.

[8] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proceedings of the IEEE*, vol. 78, No. 10, 1990, pp. 1550–1560.

[9] M. Fairbank and E. Alonso, "Value-gradient learning," in *Proceedings of the IEEE International Joint Conference on Neural Networks 2012 (IJCNN'12)*. IEEE Press, June 2012, pp. 3062–3069.

[10] M. Fairbank, D. Prokhorov, and E. Alonso, "Approximating optimal control with value gradient learning," in *Reinforcement Learning and Approximate Dynamic Programming for Feedback Control*, F. Lewis and D. Liu, Eds. New York: Wiley-IEEE Press, 2012, pp. 142–161.

[11] M. Fairbank, "Reinforcement learning by value gradients," *CoRR*, vol. abs/0803.3539, 2008. [Online]. Available: http://arxiv.org/abs/0803.3539

[12] D. Prokhorov, "Backpropagation through time and derivative adaptive critics: A common framework for comparison," in *Handbook of learning and approximate dynamic programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, 2004.

[13] M. Fairbank and E. Alonso, "The local optimality of reinforcement learning by value gradients, and its relationship to policy gradient learning," *CoRR*, vol. abs/1101.0428, 2011. [Online]. Available: http://arxiv.org/abs/1101.0428

[14] R. Munos, "Policy gradient in continuous time," *Journal of Machine Learning Research*, vol. 7, pp. 413–427, 2006.

[15] M. Fairbank, E. Alonso, and D. Prokhorov, "Simple and fast calculation of the second-order gradients for globalized dual heuristic dynamic programming in neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 10, pp. 1671–1678, October 2012.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–356, 1992.

[17] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[18] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, 1989.

[19] S. Ferrari and R. F. Stengel, "Model-based adaptive critic designs," in *Handbook of learning and approximate dynamic programming*, J. Si, A. Barto, W. Powell, and D. Wunsch, Eds. New York: Wiley-IEEE Press, 2004, pp. 65–96.

[20] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[21] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[22] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Harvard University, 1974.

[23] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 6088, pp. 533–536, 1986.

[24] E. Barnard, "Temporal-difference methods and markov models," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 2, pp. 357–365, 1993.

[25] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[26] M. Fairbank and E. Alonso, "A comparison of learning speed and ability to cope without exploration between DHP and TD(0)," in *Proceedings of the IEEE International Joint Conference on Neural Networks 2012 (IJCNN'12)*. IEEE Press, June 2012, pp. 1478–1485.

[27] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, pp. 834–846, 1983.

[28] G. G. Lendaris and C. Paintz, "Training strategies for critic and action neural networks in dual heuristic programming method," in *Proceedings of International Conference on Neural Networks, Houston*, 1997.

[29] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," Center for Cognitive and Neural Studies (Coneural), Str. Saturn 24, 400504 Cluj-Napoca, Romania, Tech. Rep., 2007.